5780_Prelab_01

Shem Snow

January 2024

1 :

The M0 processor on the STM32F072R8 has 16 kilobytes of static RAM and 128 kilobytes of flash memory.

The STM32F0 family has only this much combined physical memory, program storage, and mapped peripheral registers; however, they have a full 32-bit (4GB) address space. Therefore, huge blocks of unimplemented addresses separate each functional region of the address space (program storage, physical memory, and peripheral registers). These gaps in the address space ensure that each type of memory has a unique-bit-pattern and provides for future expansion when developing new chips.

2 :

HAL stands for **Hardware Abstraction Layer**. On our discovery board, the Device HAL has files that define bindings between peripherals on the chip and the user application. The result is a C/C++ API the user can use to control peripherals on the PCB (such as timing and communication).

3 :

Manually creating a project for an STM32F0 processor requires a significant amount of configuration which we can bypass by **using the STM32CubeMX utility to graphically configure the project parameters** and generate a ready-to-use Vision project.

4 :

Unlike in devices with an operating system, **bare-metal programs occur without the existance of any 'supervisory'/background processes such as error handling**, system monitorying, garbage collection, memory management, scheduling, etc... **Therefore the 'return' function is undefined**.

There is nothing that will 'catch' and exception or handle an error. There is no place to "return" to.

5 :

Microcontrollers use specialized registers as a permanent read/write location for peripherals. The memory structure abstracts them as their own space in memory as shown in the table.

Core peripherals
Peripherals
SRAM
Program Code

$\uparrow address$

The peripheral registers are at a higher address than SRAM.

6 :

The lab manual says this:

1. **(STM32F072RBT6 Datasheet) DM00090510.pdf**
   - The chip datasheet provides device-specific details for the processor; this includes pin connections for available chip packages and a list of available peripherals.
2. **(Programming & Core Manual) DM00051352.pdf**
   - The core programming manual provides information on the ARM-core peripherals as well as the assembly instruction set; it is generic to all of the processors within the STM32F0 family.
3. **(Peripheral Manual) DM00031936.pdf**
   - The peripheral reference manual contains detailed information on all peripherals available within an STM32F0 device; however, not all STM32F0 devices contain every peripheral! The chip datasheet is necessary to determine which peripherals are available for use.
4. **(Discovery Board Manual) DM00099401.pdf**
   - The Discovery board manual contains schematics and tables that show the onboard devices and connectors attached to the STM32F0; the Discovery board silkscreen also documents many device connections.

Figure 1:

However, these names are not the same as the 4 data sheets that were provided on Canvas. I think they are referring to the same ones so here's the answer I came up with when actually looking through them.

1. **RM0091 Reference manual explains the design of all 32-bit microprocessors in the TM32F0x 1, 2, and 8 families**. It talks about their capabilities such as their peripherals, the clocks they implement, the power supply compatibility information, and memory organization.

It also talks about their protocols for memory read/write protection and how the device responds to events such as: flash memory interrupts, clock recovery, interrupts. Then there is information about individual capabilities of each section/partition of the device (ACD, DAC, sensing, watchdogs, interfaces, transmitters, etc..). In short, **this is the document to read if you want to SEE IF THE MICROCONTROLLER HAS ALL THE DESIRED PERPHERALS AND WILL WORK FOR YOUR APPLICATION SO YOU CAN BUY IT**.

2. **STM32F072x8 STM32F072xB** is ~~the more useful document that I'm actually going to reference~~ much like the previously mentioned RM0091 Reference manual except only for the -8 family. It has more specific (therefore practical) information about the system and memory architecture, protocols, and limitations. When somebody reads through this they will develop a deep and direct understanding of the device. In short, **this is the document to read if you want to LEARN HOW THE MICROCONTROLLER WORKS**.

3. **PM0215 Programming manual** is the manual for specifically the M0 processor within the family. **It covers topics from a software application approach** such as: defining the instruction set and data types, explaining all the registers and their read/write restrictions, providing an abstracted memory architecture that goes from 0x00000000-0xFFFFFFFF and shows how each region is reserved for different peripherals or memory, exception triggering and handling, etc... In short, **this is the document to read if you want to LEARN HOW TO USE THE MICROCONTROLLER**.

4. **UM1690 User manual is the user manual for our discovery kit**. It talks about what the PCB is expected to do and how it is partitioned into different regions for each function. It discusses the purpose of individual components such as diodes that protect the board from power supplies and resistors that may effect application results and should be de-soldered for certain applications. It talks about how there is a second processor on the PCB which is permanently hard-coded because its purpose is solely to interface with the external system that writes code in order to put the code into the actual M0 processor. It talks about all the pins and how to (NOT) use them.

7 :

STM32F0 devices do not recognize inputs/outputs on a chip by physical pin numbering because different chip packages have differing numbers of pins, and the pin ordering between them is inconsistent; GPIO pins are instead labeled with a port name (PA0 for example) which describes where to go to configure

it. The chip's datasheet contains a table mapping GPIO pin names to physical pin numbers on the specific chip package.

8 :

For devices from ST Electronics, the header file that defines names for the peripheral registers are in the "CMSIS Cortex-M0 Device Peripheral Access Layer Header Files", also known as stm32f0xx.h where "xx" specifies each device's sub-family. I can locate these files manually by searching for them within the project directory under: "Drivers\CMSIS\Device\ST\STM32F0xx\Include".

9 :

To set only some of a register's bits TO ONE and leave the others untouched, use the bit-wise OR operation of that register with another register that contains all zeros in the positions that do not need to be set and contains ones in the positions that do need to be set.

10 :

Since synchronous circuits don't operate without a clock signal, unused peripherals can be "turned-off" by disabling their clock signals. This means we have to connect the microcontroller's clock signal into each peripheral we want to use.

The STM32F0 family has a dedicated peripheral called the Reset and Clock Control (RCC) which enables us to do that. There are three registers (one for each BUS) that enable the clocks in all peripherals except the ARM core itself.

11 :

The HAL library uses the SysTick timer peripheral, which raises a system signal at a configurable periodic rate, to trigger a periodic tasks such as updating a global system time variable. The HAL_Delay() function, which halts execution for a specified number of millisecond, is in that library.

12 :

While most of the C-standard variable types work properly on embedded systems, there is always the possibility that they are not the size (number of bits) as expected. On some systems, integers are 16 bits and on others they are 32.

Additionally, **many embedded devices, including our discovery board, do not have hardware support for floating-point arithmetic** and must emulate it with large and slow code libraries.